Adrià Ciurana Lanau

Udacity:

Project 2: Tennis

Project description:

In this project it is proposed to solve a problem where there are two agents that collaborate and compete. Specifically the environment is a game of **ping pong** where each agent should try to avoid losing. +0.1 if it hits the ball and -0.01 if an agent lets a ball hit the ground or hits the ball out of bounds.

Although this problem is multi agent, it has the particularity that they do not have the need to know the other agent's information and act accordingly. Therefore you can use a conventional DDPG, which works with multiple agents and a shared replay buffer that makes it learn a single **DDPG network.**

In the first instance, we should try to formalize the problem proposed in one of RL.

Variable name	Description
state	A set of 8 sensors that describe the information of the current state.
actions	A set of 2 continue actions with a range between -1 to 1.
reward	+0.1 if the agent hit the ball or -0.01 if the agent lets a ball hit the ground or hits the ball of of bounds.

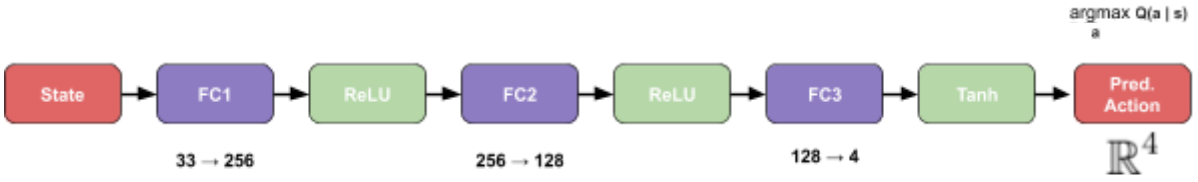
Once the problem is defined, we must remember that an RL problem is simply a **maximization problem**. In contrast the previous project, in this case we use a maximization problem that combine: Actor-Network and Critic-Network.
The problem can be resolved exactly as the previous project, the small difference is the great instability to depend on the state of the environment of another agent. With this you can ask us a question: **What would happen both agents have not yet learned and never hit the ball?**

Actor-Critic networks uses a combination of Policy based (Actor) and Value based (Critic). To minimize the variance of the Actor and the bias of the Critic. This approximation tries to get the best of both worlds.

In our case, we will use DDPG with multiple **unsynchronized** agents. This means that we have 2 agents player ping-pong and storing experiences in a common buffer and a single network that obtains the experiences to be learned.

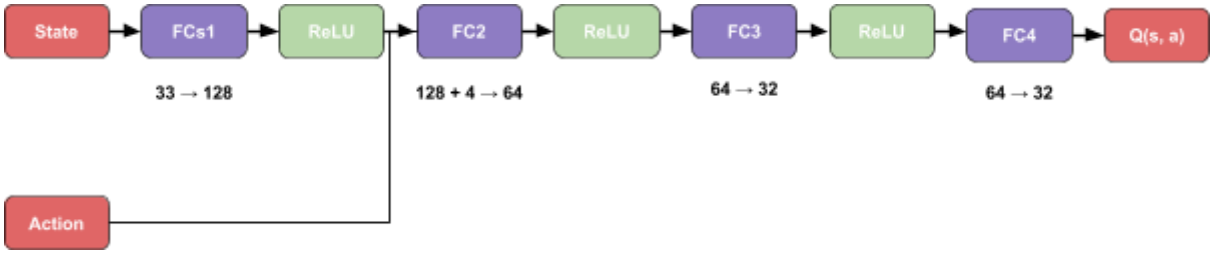
Actor Network:

As the problem is continuous, it is different from the initial concept of the policy based network. Because how could we obtain a function of distribution of continuous values as output? It would be somewhat difficult to work with, for that reason the output is modified and instead of being an output based on a distribution, this becomes a **deterministic output** that is the action that maximizes the probability.



Critic Network:

In this case, as the function is continuous, we must introduce the action within the critic network. The output is only a value that estimates Q (s, a).



Loss definition:

Any optimization problem has a loss function associated with it. In this case, we have defined a loss based DDPG paper. This loss is composed of two phases: critic and actor.

Critic Loss:

$$q(s', a') = Critic(Actor(s'), a')$$
$$q(s, a) = \bar{Critic}(s, a)$$
$$\mathcal{L}_{Critic} = ||\bar{q}(s, a) - [r + \gamma \cdot q(s', a')]\,||^2$$

Actor Loss:

$$\mathcal{L}_{Actor} = -\bar{Critic}(s, \bar{Actor}(s))$$

Agent step:

In this case the learning process is divided into two parts:

1. **Learning of the critical network:** The best next action is predicted using the actor network (**fixed**) and then the critical loss function is minimized.
2. **Learning of the actor network:** The current action is predicted and it is treated as having the highest Q (s, a) possible (**using critic as fixed**).

Subsequently, a smooth update is used to transition the learning of the new networks to the fixed networks. Unlike **DQN**, this transition is very smooth.

When the agent acts with the environment and is in the process of learning, noise is added to the action. Since the action is between -1 and 1, it is quite easy and it is enough to add **Gaussian noise**. This allows to improve the **exploration** of the agent.

To work with multiple agents at the same time, a replay buffer and a global neural network have been used. While agents asynchronously have been filling it with experiences.

Hyperparameters:

The hyperparameters have been based on the DDPG-Pendulum project, only modifying the **batch size to 1024** to allow a better generalization (since the problem is much larger).

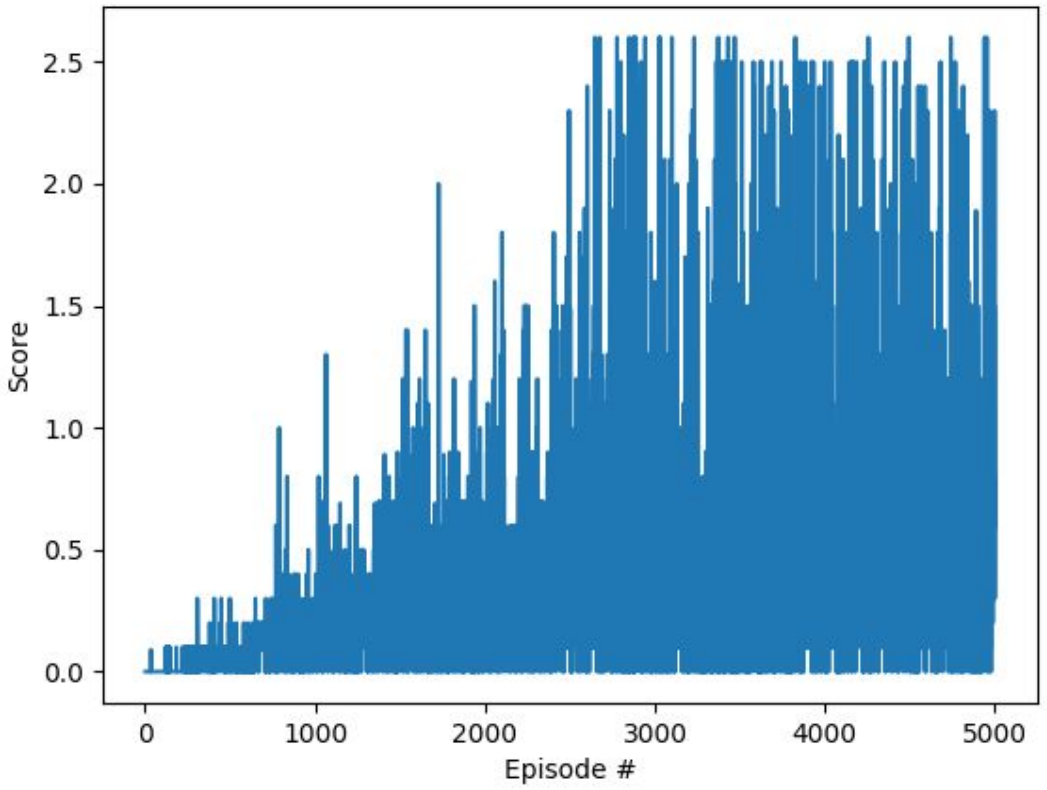
In addition, the **experience buffer is increased to $1x10^9$.**

The **discount factor have fixed to 1.**

In order to reuse the experiences from the buffer and make more stable the learning process, **we learn the network 50 times for each 10 steps of the agents**.

Results:

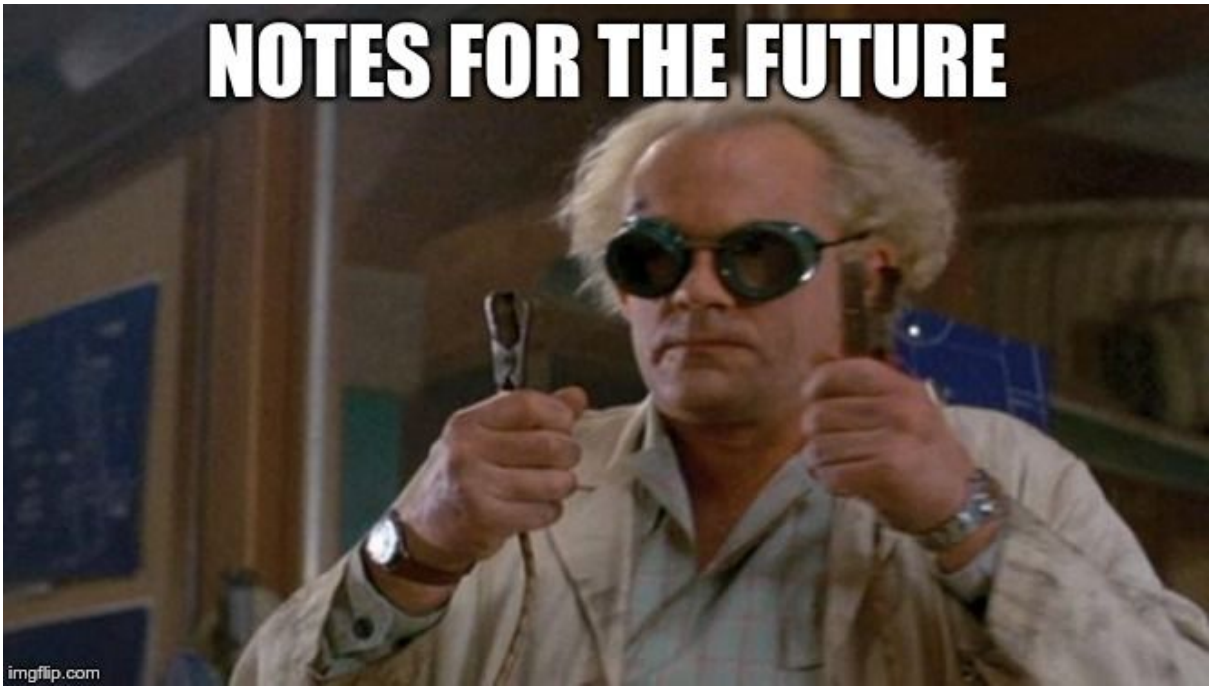
Below are the results obtained during the episodes. Learning has been stopped when the agents do 5000 episodes.



Results (last episodes):

```
Episode: 4996
      - Current Score: 1.500000 (+/- 0.005000)
      - 100 Avg Score: 0.628100 (+/- 0.623393)
================================================
Episode: 4997
      - Current Score: 0.590000 (+/- 0.045000)
      - 100 Avg Score: 0.633000 (+/- 0.621145)
================================================
Episode: 4998
      - Current Score: 2.300000 (+/- 0.055000)
      - 100 Avg Score: 0.653000 (+/- 0.641951)
================================================
Episode: 4999
```

```
      - Current Score: 0.300000 (+/- 0.005000)
      - 100 Avg Score: 0.655000 (+/- 0.640535)
================================================
Episode: 5000
      - Current Score: 0.800000 (+/- 0.055000)
      - 100 Avg Score: 0.648000 (+/- 0.635064)
```



In the future…

- Try another mechanism of Multi-agent learning.
- Test new hyperparameters.
- It would have been interesting to try other network architectures, together with new RL ideas (for example: A2C, A3C, ...).
- Try to solve multi-agent with a more stable method.
- Apply data augmentation to make a mirror in the experiences and that the experiences of an agent were useful for the other one.